EVIM

ENGLISH VALIDATION
&
INTERNSHIP MANAGEMENT

Object Design Document

EVIM - English Validation & Internship Management

Riferimento	
Versione	1.2
Data	05/01/2020
Destinatario	Top Management
Presentato da	Simone Auriemma - Michele Duraccio – Antonio Giano – Simona Grieco – Emilio Schiavo – Maria Concetta Schiavone – Nicola Sisti – Vincenzo Colacicco
Approvato da	Edoardo Carpentiero – Attilio Della Greca

Team Composition

Ruolo	Nome	Posizione	Contatti
Top Manager	Filomena Ferrucci	Rappresentante del cliente	f.ferrucci@unisa.it
Project Manager	Edoardo Carpentiero	Project Manager	e.carpentiero1@studenti.unisa.it
Project Manager	Attilio Della Greca	Project Manager	a.dellagreca5@studenti.unisa.it
Team Member	Simone Auriemma	Team Member	s.auriemma5@studenti.unisa.it
Team Member	Vincenzo Colacicco	Team Member	v.colacicco1@studenti.unisa.it
Team Member	Duraccio Michele	Team Member	m.duraccio3@studenti.unisa.it
Team Member	Giano Antonio	Team Member	a.giano1@studenti.unisa.it
Team Member	Grieco Simona	Team Member	s.grieco13@studenti.unisa.it
Team Member	Emilio Schiavo	Team Member	e.schiavo8@studenti.unisa.it
Team Member	Maria Concetta Schiavone	Team Member	m.schiavone29@studenti.unisa.it
Team Member	Nicola Sisti	Team Member	n.sisti1@studenti.unisa.it

Sommario

Revision History

Data	Versione	Descrizione	Autori
02/12/2019	0.1	Impostazione documento	Team Member
07/12/2019	0.2	Stesura Introduzione	Vincenzo Colacicco
08/12/2019	0.3	Stesura Linee guida per l'implementazione e Componenti off-the-shelf	Antonio Giano Vincenzo Colacicco
12/12/2019	0.4	Stesura Design Pattern	Simone Auriemma
13/12/2019	0.5	Stesura Package	Antonio Giano Simone Auriemma Emilio Schiavo
14/12/2019	0.6	Stesura Class Interface	Antonio Giano Simone Auriemma Emilio Schiavo
15/12/2019	0.7	Revisione interna	Team Member
16/12/2019	1.0	Revisione	Edoardo Carpentiero Attilio Della Greca
30/12/2019	1.1	Revisione class interfaces	Emilio Schiavo
05/01/2020	1.2	Revisione e impaginazione definitiva	Emilio Schiavo

1. Introduzione

2. Il presente documento viene redatto, dopo la stesura del Requirements Analisys Document e del System Design Document, allo scopo di offrire una panoramica del sistema, collegando con precisione tutte le informazioni raccolte nelle suddette fasi. In particolare, si vogliono illustrare le interfacce delle classi, le operazioni supportate, i tipi di dati trattati, i parametri delle procedure e i signature dei sottosistemi definiti nel System Design Document.

2.1 Object Design Trade-offs

- **Comprensibilità vs Tempo**

Un aspetto fondamentale per assicurare la buona qualità di un progetto software è la comprensibilità del codice: esso dovrebbe sempre essere di facile interpretazione, non solo durante la fase di testing ma anche per eventuali soggetti terzi che non hanno mai collaborato alla realizzazione del sistema. Ai fini di ciò è stato utilizzato l'applicativo Javadoc, incluso nel Java Development Kit della Sun Microsystem ed un tool di sviluppo denominato "CheckStyle", che permette l'automazione del processo di controllo della formattazione di codice sorgente.
Nonostante questo, a causa del ridotto tempo a disposizione, si preferisce sempre dare precedenza ai tempi di rilascio schedulati, anche a costo di avere una comprensibilità minore, pur avendo almeno il 95% di codice senza warning rilevati da CheckStyle. Essendo il progetto della dimensione di circa 20.000 linee di codice, il numero di warning ammessi è pari a 1000.

- **Prestazioni vs Costi**

Nonostante il budget disponibile sia allocato quasi interamente in risorse umane, un rapporto ottimale tra efficienza e performance sarà assicurato dall'impiego dei migliori strumenti open source attualmente disponibili.

- **Sicurezza vs Efficienza**

La gestione della sicurezza del sistema è affidata interamente al modulo di login, che richiede l'inserimento di un indirizzo e-mail ed una password. Il sistema controlla nella banca dati se tali informazioni sono presenti quindi consente l'accesso all'utente, naturalmente con diritti d'accesso differenti a seconda del ruolo dello stesso. Questo metodo consente un buon equilibrio tra efficienza (perché mantiene il sistema leggero) e grado di sicurezza, rientrando nel budget a disposizione per la realizzazione del progetto.

- **Tempo di risposta vs Spazio di memoria**

È stato scelto l'utilizzo di un database relazionale in modo da trarre vantaggio dalle seguenti caratteristiche:

- Consistenza dei dati
- Bassa latenza nella risposta del filesystem

- Accesso rapido e simultaneo dei dati

Un database relazionale ha tuttavia l'inconveniente di utilizzare una quantità maggiore di spazio su disco, ciò può generare degli inconvenienti nella gestione di grosse quantità di dati.

- **Interfaccia vs Usabilità**

Allo scopo di una fruizione semplice ed immediata del sistema, le interfacce saranno dotate di uno stile grafico leggero ed una disposizione intuitiva dei comandi, utilizzando parole chiave d'uso comune, ciò al fine di semplificare l'esperienza dell'utente finale ed incrementare il suo grado di soddisfazione. L'usabilità del sistema sarà garantita da un test di usabilità eseguito al termine dello sviluppo.

2.2 Definizioni acronimi e abbreviazioni

- **Design Pattern**: modello logico utilizzato in fase di risoluzione di problematiche ricorrenti che possono sorgere durante la progettazione e/o sviluppo del software.
- **Class Diagram**: Diagramma descrivente la conformazione di un sistema, esso presenta le classi con i relativi attributi, le operazioni effettuabili e quali relazioni vi sono fra gli oggetti.
- **MVC**: acronimo di Model-View-Controller, è un modello utilizzato per lo sviluppo di sistemi software basati sulla programmazione ad oggetti. Segue descrizione dettagliata al paragrafo 3.1 del presente documento.
- **Package**: pacchetto che include classi e correlazioni con interfacce.
- **Package Model**: pacchetto che include la parte Model (interazione tra applicazione e database) del modello MVC.
- **Package Docs:** pacchetto che include la parte View (prospettiva che permette all'utente l'interazione col sistema) del modello MVC.
- **Package Routes**: pacchetto che include la parte Controller (elaborazione delle richieste dell'utente e dialogo con la parte Model) del modello MVC.
- **GPL**: acronimo di General Public License, licenza che consente a chiunque di utilizzare, modificare e redistribuire il software rilasciato tramite la medesima.

2.3 Definizioni acronimi e abbreviazioni

- Kathy Schwalbe, "Information Technology Project Management", International Edition 7E, Cengage Learning, 2014;
- Bernd Bruegge, Allen H. Dutoit, "Object-Oriented Software Engineering Using UML, Patterns and Java", Third Ed., Pearson, 2010;
- Sommerville, "Software Engineering", Addison Wesley;
- PMBOK ® Guide and Software Extention to the PMBOK® Guide, Fifth Ed., Project Management Institute, 2013;

Documentazione integrativa di progetto:

- EVIM_RAD_Vers1.6;
- EVIM_SDD_Vers1.2;

3. Linee guida per l'implementazione

Nell'implementare il sistema, i programmatori dovranno attenersi alle linee di guide mostrati in seguito:

3.1 Organizzazione dei file

I file vengono organizzati secondo il criterio MVC la cui descrizione è spiegata nella sezione **Pattern MVC**:

- File di tipo **view**: file di questo tipo devono essere collocati all'interno della cartella WebContent/WEB-INF, creata dall'IDE Eclipse, in modo da evitare accessi non autorizzati da parte degli utenti. Il nome di un file deve essere scritto con una parola della pagina corrispondente combinata alla parola raffigurante un qualunque elemento dell'interfaccia grafica.
    - Esempio: compilazioneRegistroForm.jsp
- File di tipo **model**: file di questo tipo devono essere collocati in una cartella creata dallo sviluppatore denominata "Model" e situata all'interno della cartella creata da Eclipse denominata "src". Il nome di un file deve essere scritto con una parola che rappresenti il modello del dominio del problema;
    - Esempio: studente.java
- File di tipo **control**: file di questo tipo devono essere collocati in una cartella creata dallo sviluppatore denominata "control" all'interno della cartella creata da Eclipse denominata "src". Il nome di un file deve essere scritto con una parola che rappresenti la pagina corrispondente con aggiunta come desinenza "Servlet";
    - Esempio: formServlet.java
- Filo di tipo **CSS**: file di questo tipo devono essere collocati in una cartella creata dallo sviluppatore denominata "CSS" all'interno della cartella creata da Eclipse denominata WebContent. Il nome di un file deve inoltre raffigurare la pagina con cui si andrà a configurare lo stile.
    - Esempio: stiliListaTirocini.css

3.2 Classi Java

Ci si atterrà allo standard Java Enterprise Edition nella scrittura del codice. In particolar modo ci si focalizza sulle seguenti regole:

1. Nomi delle variabili, classi e metodi seguono la notazione lowerCamelCase, ovvero la prima lettera della prima parola in minuscolo e la prima lettera di ogni altra parola in maiuscolo.
2. È consigliato commentare con la tecnica Javadoc ogni classe e ogni metodo in modo da facilitare la comprensione. Il commento javadoc deve essere scritto prima della classe e di ogni metodo. È possibile commentare un'istruzione singola o composta nel caso in cui tale istruzione risulta complessa da capire a prima vista. Viene scritta prima dell'implementazione di tale istruzione singola o composta.
3. Nessuno spazio inserito tra il nome del metodo e la parentesi tonda "(" dove quest'ultima indica la lista dei parametri.
4. La parentesi graffa "{" deve essere alla fine della linea di dichiarazione.
5. La parentesi graffa "}" deve essere nella riga successiva allo stesso livello di indentazione della linea di dichiarazione della classe o interfaccia.
6. Le istruzioni che sono incorporati all'interno di un blocco di istruzione dovranno essere indentate di un'unità.
7. I metodi devono essere nominati come verbi mentre la classe e le variabili devono essere nominati come sostantivi riguardanti al dominio del problema.

Esempio:

```
/**
        Scopo della classe
*/
public class Myclass{
        /** aggiunge qualcosa*/
        public void add(){
                //codice da implementare
        }
}
```

3.3 HTML

Le pagine HTML, in formato statico o dinamico, devono seguire le regole previste dallo standard HTML5, di cui alcune opzionali:

1. Un file HTML deve iniziare con il tag di apertura <html> e tag di chiusura </html>.
2. Ai fini di una migliore leggibilità, i tag di apertura e chiusura devono essere indentati.
3. Ogni tag deve essere indentato di un'unità rispetto al tag che lo contiene.
4. Ogni tag di chiusura deve essere indentato allo stesso livello del tag di apertura.
5. E' possibile commentare un blocco di codice per migliorarne la comprensibilità, un commento è espresso in questo modo: <!-- comment --> e viene scritto prima della scrittura di un blocco di codice corrispondente.

Esempio:

```
<html>
        <head>
            <!-- codice da implementare-->
        </head>
        <body>
        <!-- codice da implementare-->
        </body>
</html>
```

3.4 JSP: Java Server Page

Le pagine JSP devono seguire regole previste dallo standard HTML.
Segue la stessa codifica di Java con alcune aggiunte:

1. Il tag di apertura (<%) è collocato all'inizio della riga.
2. Il tag di chiusura (%>) è collocato alla fine della riga comprendente il tag di apertura.
3. Nel caso in cui un blocco di codice JSP è formato da più di una istruzione, il tag di apertura deve essere collocato nella riga precedente del blocco di codice JSP e il tag di chiusura nella riga successiva al relativo blocco di codice JSP.

Esempio:

```
<% out.print("ciao")%>
<%
    for (String parametri: temp)
    out.print(temp)
%>
```

3.5 JavaScript

Segue le stesse convenzioni espresse nella linea guida della sezione *2.2 Classi Java* con alcune aggiunte:

- È possibile inserire codice all'interno del tag corrispondente qualora lo script da scrivere non sia eccessivamente ingombrante.
  - Esempio:

```
<button
onclick="document.getElementById("demo").innerHTML=Date
()"> </button>
```

- Se lo script risulta ingombrante, si inserisce nella sezione apposita, ovvero all'interno dei tag <script> </script>.
  - Esempio:

```
<script>
        function displayDate() {
        document.getElementById("demo").innerHTML = Date();
    }
</script>
```

3.6 CSS: Cascade Style Sheets

I file di tipo CSS devono seguire le seguenti regole:

1. tutti gli stili *non inline* devono essere collocati su un file dedicato.
2. Nei file con gli stili *non inline*, ogni selettore deve essere indentato a livello 0 ed ogni proprietà deve essere indentata di un'unità in più rispetto al selettore che lo contiene.
3. L'ultimo selettore di ogni regola è seguito da una parentesi graffa aperta "{".
4. Ogni regola è terminata da una parentesi graffa chiusa "}", inserita da sola su una riga successiva.
5. Ai fini di comprendere meglio una regola CSS, è possibile inserire un commento qualora una regola non fosse chiara, va collocato nella riga precedente della relativa regola CSS.

Esempio:

```
/* stili per i messaggi output*/
out{
    color:red;
    position:fixed;
}
```

3.7 Database SQL

I nomi delle tabelle devono rispettare tali regole:

1. Seguono la convenzione UpperCamelCase ovvero la prima lettera di ogni parola deve essere maiuscola.
2. Il nome di una tabella deve essere un sostantivo singolare preso dal dominio del problema.

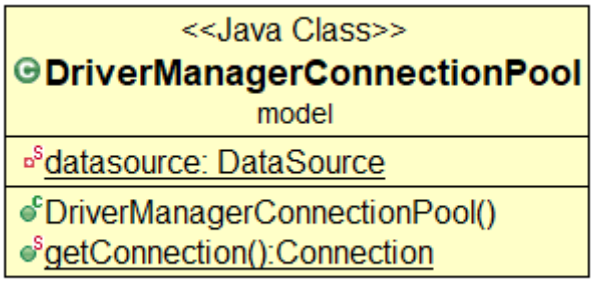
I nomi dei campi devono seguire le seguenti linee guida:

1. Nomi con lunghezza minore di 3 vanno scritti con tutti i caratteri in maiuscolo, nomi con lunghezza maggiore di 3 vanno scritti con la prima lettera della parola in maiuscolo.
2. Se il nome è formato da due parole, è necessario separarli con l'underscore "_".
3. Il nome di un campo deve essere un sostantivo singolare preso dal dominio del problema.

4. Design Pattern

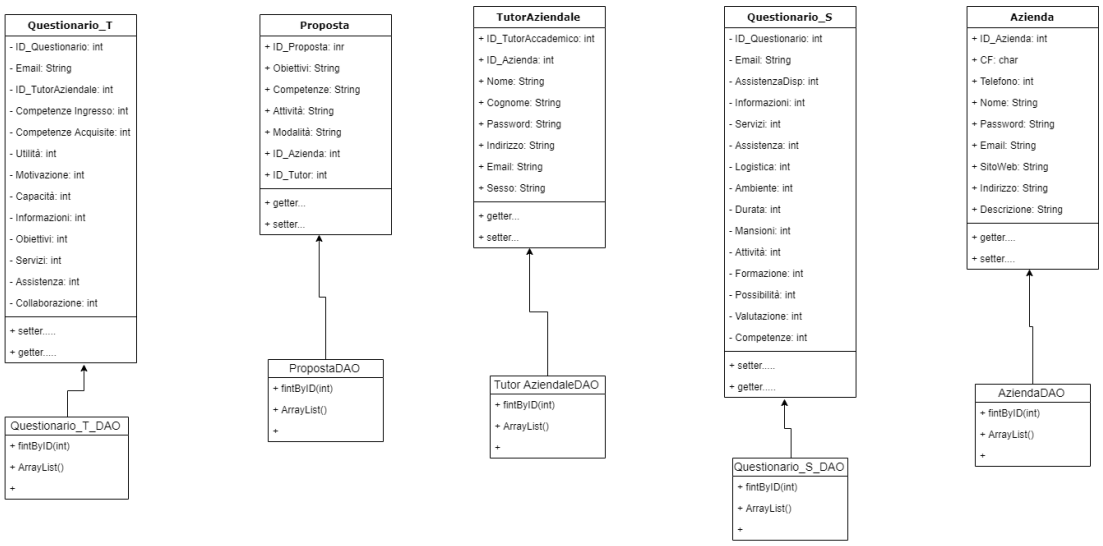
4.1 Singleton

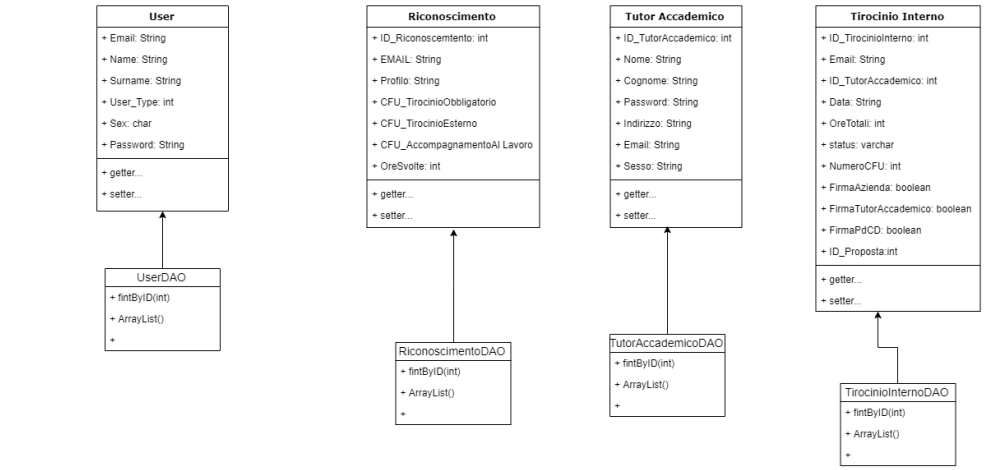
È stata progettata, per motivi di efficienza, una classe **DriverManagerConnectionPool**, la quale permette di stabilire una connessione tra l'utente e il sistema, in questo modo la connessione verrà effettuata solamente una volta, quando si accede ad una qualsiasi pagina. La connessione rimarrà attiva se l'utente sarà attivo, mentre cadrà automaticamente se dopo un periodo di tempo stabilito l'utente non effettua altre azioni con il sistema. Perdendo lo stato della sessione attuale, sarà necessario effettuare nuovamente il login per accedere alle funzionalità non accessibili agli utenti ospiti.
Questa classe verrà resa singleton, quindi la sua istanza sarà in comune con tutte le classi che interagiscono con essa.

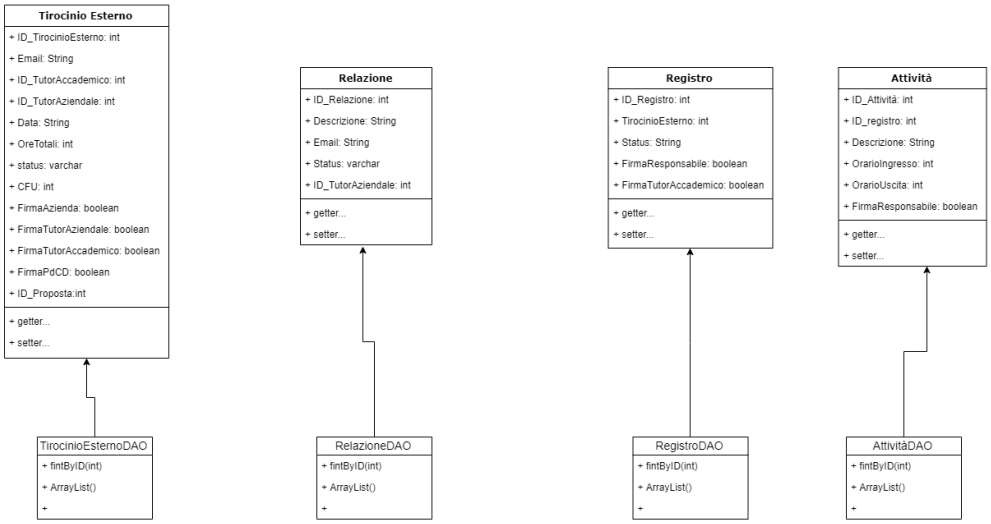


4.2 Design Pattern DAO

La gestione del Model è stata fatta seguendo il pattern architetturale DAO, uno degli Standard J2EE Design Patterns, che prevede la creazione di due classi per ogni entità, una entityDAO che prevede l'implementazione dei metodi per le operazioni CRUD ed una entityClass, che è l'implementazione stessa in linguaggio Java dell'entità presente sul db.


Questionario_T
- ID_Questionario: int
- Email: String
- ID_TutorAziendale: int
- Competenze Ingresso: int
- Competenze Acquisite: int
- Utilità: int
- Motivazione: int
- Capacità: int
- Informazioni: int
- Obiettivi: int
- Servizi: int
- Assistenza: int
- Collaborazione: int
+ setter.....
+ getter.....
Questionario_T_DAO
+ fintByID(int)
+ ArrayList()
+
Proposta
+ ID_Proposta: inr
+ Obiettivi: String
+ Competenze: String
+ Attività: String
+ Modalità: String
+ ID_Azienda: int
+ ID_Tutor: int
+ getter...
+ setter...
PropostaDAO
+ fintByID(int)
+ ArrayList()
+
TutorAziendale
+ ID_TutorAccademico: int
+ ID_Azienda: int
+ Nome: String
+ Cognome: String
+ Password: String
+ Indirizzo: String
+ Email: String
+ Sesso: String
+ getter...
+ setter...
Tutor AziendaleDAO
+ fintByID(int)
+ ArrayList()
+
Questionario_S
- ID_Questionario: int
- Email: String
- AssistenzaDisp: int
- Informazioni: int
- Servizi: int
- Assistenza: int
- Logistica: int
- Ambiente: int
- Durata: int
- Mansioni: int
- Attività: int
- Formazione: int
- Possibilità: int
- Valutazione: int
- Competenze: int
+ setter.....
+ getter.....
Questionario_S_DAO
+ fintByID(int)
+ ArrayList()
+
Azienda
+ ID_Azienda: int
+ CF: char
+ Telefono: int
+ Nome: String
+ Password: String
+ Email: String
+ SitoWeb: String
+ Indirizzo: String
+ Descrizione: String
+ getter....
+ setter....
AziendaDAO
+ fintByID(int)
+ ArrayList()
+


User
+ Email: String
+ Name: String
+ Surname: String
+ User_Type: int
+ Sex: char
+ Password: String
+ getter...
+ setter...
UserDAO
+ fintByID(int)
+ ArrayList()
+
Riconoscimento
+ ID_Riconoscemtento: int
+ EMAIL: String
+ Profilo: String
+ CFU_TirocinioObbligatorio
+ CFU_TirocinioEsterno
+ CFU_AccompagnamentoAl Lavoro
+ OreSvolte: int
+ getter...
+ setter...
RiconoscimentoDAO
+ fintByID(int)
+ ArrayList()
+
Tutor Accademico
+ ID_TutorAccademico: int
+ Nome: String
+ Cognome: String
+ Password: String
+ Indirizzo: String
+ Email: String
+ Sesso: String
+ getter...
+ setter...
TutorAccademicoDAO
+ fintByID(int)
+ ArrayList()
+
Tirocinio Interno
+ ID_TirocinioInterno: int
+ Email: String
+ ID_TutorAccademico: int
+ Data: String
+ OreTotali: int
+ status: varchar
+ NumeroCFU: int
+ FirmaAzienda: boolean
+ FirmaTutorAccademico: boolean
+ FirmaPdCD: boolean
+ ID_Proposta:int
+ getter...
+ setter...
TirocinioInternoDAO
+ fintByID(int)
+ ArrayList()
+

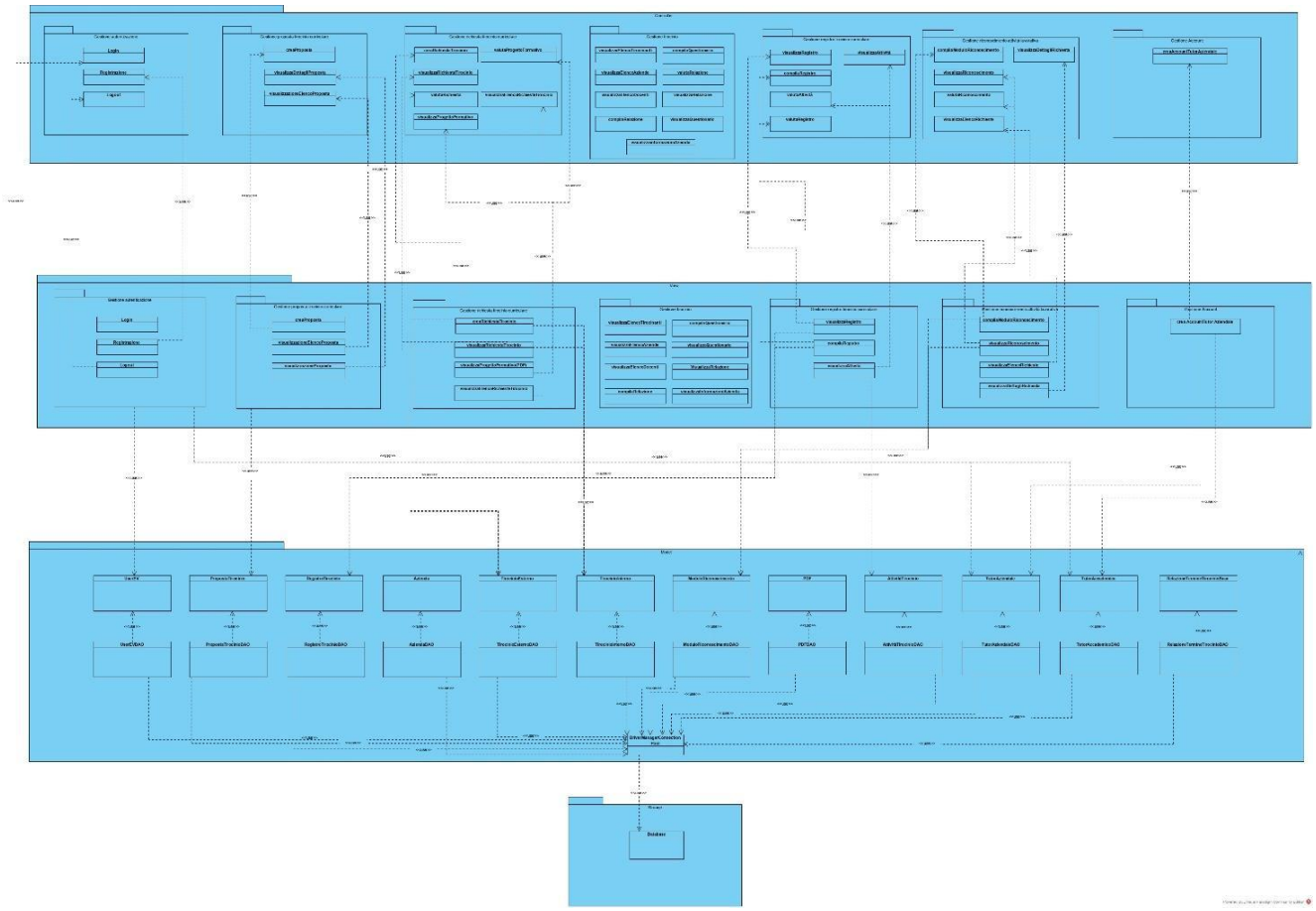

Tirocinio Esterno
+ ID_TirocinioEsterno: int
+ Email: String
+ ID_TutorAccademico: int
+ ID_TutorAziendale: int
+ Data: String
+ OreTotali: int
+ status: varchar
+ CFU: int
+ FirmaAzienda: boolean
+ FirmaTutorAziendale: boolean
+ FirmaTutorAccademico: boolean
+ FirmaPdCD: boolean
+ ID_Proposta:int
+ getter...
+ setter...
TirocinioEsternoDAO
+ fintByID(int)
+ ArrayList()
+
Relazione
+ ID_Relazione: int
+ Descrizione: String
+ Email: String
+ Status: varchar
+ ID_TutorAziendale: int
+ getter...
+ setter...
RelazioneDAO
+ fintByID(int)
+ ArrayList()
+
Registro
+ ID_Registro: int
+ TirocinioEsterno: int
+ Status: String
+ FirmaResponsabile: boolean
+ FirmaTutorAccademico: boolean
+ getter...
+ setter...
RegistroDAO
+ fintByID(int)
+ ArrayList()
+
Attività
+ ID_Attività: int
+ ID_registro: int
+ Descrizione: String
+ OrarioIngresso: int
+ OrarioUscita: int
+ FirmaResponsabile: boolean
+ getter...
+ setter...
AttivitàDAO
+ fintByID(int)
+ ArrayList()
+

5. Componenti Off-the-shelf

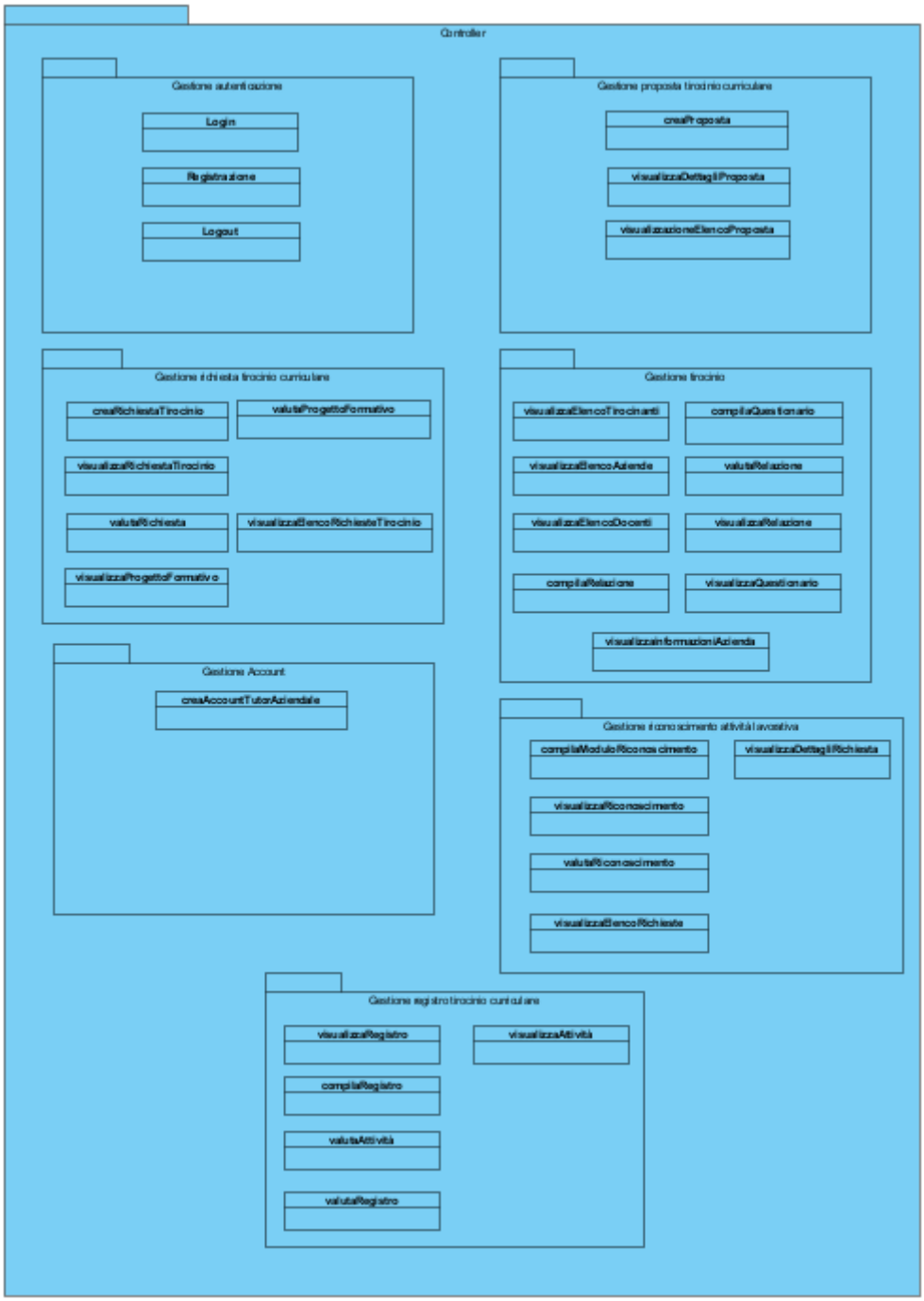
Di seguito vengono illustrati i componenti e le librerie esterne di pubblico dominio utilizzati per lo sviluppo del sistema:

- **MySQL Connector/J**: È un driver che consente ad un programma scritto in linguaggio Java la connessione e l'interazione con una banca dati creata e gestita con MySQL.
- **GearHost**: Provider per host di databases online, è stato utilizzato per avere un database online condiviso da tutti i developer.
- **JavaScript**: Si tratta di un linguaggio di automazione orientato agli oggetti, utilizzato nella creazione di applicazioni e siti web. A differenza di altri linguaggi definiti standalone come C o Java coi quali condivide parte della sintassi, esso si integra all'interno di un altro programma preesistente e il suo codice non viene compilato ma direttamente interpretato. Utile per la scrittura di piccole funzioni integrate nelle pagine web per compiere azioni altrimenti non possibili con il solo HTML statico, come ad esempio il controllo di valori inseriti in un modulo.
- **JSTL**: Acronimo di JavaServer Pages Standard Tag Library, è una libreria della suite di sviluppo per applicazioni web Java EE che consente, tramite un insieme di tag HTML programmati in linguaggio Java che estende la tecnologia JSP, di poter gestire ed automatizzare task ricorrenti come la manipolazione di dati in formato XML, l'accesso a una banca dati o l'internazionalizzazione (localizzazione) del software.
- **Fpdf**: Utilizzata come estensione del linguaggio PHP, questa libreria consente in maniera molto semplice ma al tempo stesso precisa la produzione di documenti in formato PDF. Offre supporto ai più diffusi formati di file contenenti immagini, alla gestione di collegamenti ipertestuali, alla possibilità di scelta del formato delle pagine e dei margini, alla cifratura e molto altro.
- **Bootstrap**: Si tratta di una collezione di strumenti open source realizzati appositamente per la creazione di applicazioni e siti web. Include diversi modelli di progettazione sia per quanto riguarda l'impaginazione del testo sia per l'interfaccia (bottoni, form). Il tutto è basato sui linguaggi HTML e CSS.
- **AJAX**: Acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo di applicazioni web multipiattaforma, compatibile con diverse architetture e sistemi operativi. Essa fa uso di una combinazione di HTML e CSS per ciò che concerne la formattazione e lo stile del testo, JavaScript per l'interazione e la presentazione delle informazioni e XML come formato di scambio dei dati. AJAX funziona in maniera asincrona: i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina visualizzata in un determinato istante.
- **JQuery**: Libreria di JavaScript destinata alle applicazioni web. Consente di semplificare la selezione, la manipolazione e la gestione degli eventi nelle pagine scritte in HTML, funge da supporto a Bootstrap ed è fondamentale per l'implementazione di AJAX.

6. Package



6.1 Package Controller



6.1.1 Gestione Autenticazione

Classe	Descrizione
Login	Controller che permette l'accesso da parte di un utente registrato nel sistema EVIM.
Logout	Controller che permette l'uscita di un utente dal sistema EVIM.
Registrazione	Controller che gestisce la registrazione di un utente non registrato nella piattaforma EVIM

6.1.2 Gestione Proposta Tirocinio Curriculare

Classe	Descrizione
creaProposta	Controller che gestisce la creazione di una proposta di studia da parte del tutor accademico e dall'azienda.
visualizzaPropriaProposta	Controller che permette la visualizzazione delle proprie proposte di tirocinio nel caso del tutor accademico, tutor aziendale. Mentre nel caso del referente aziendale, permette la visualizzazione di tutte le proposte di tirocinio da lui pubblicate.
visualizzaProposta	Controller che permette la visualizzazione dei dettagli di una proposta specifica.

6.1.3 Gestione Richiesta Tirocinio Curriculare

Classe	Descrizione
creaRichiestaTirocinio	Controller che gestisce la creazione di una richiesta di tirocinio curriculare di uno studente che intende iniziare il tirocinio.
visualizzaRichiestaTirocinio	Controller che permette la visualizzazione delle richieste di tirocinio inoltrate dagli studenti nel caso del tutor accademico e del referente aziendale. Nel caso dello studente, permette la visualizzazione le

	proprie richieste di tirocinio inoltrate.
valutaRichiesta	Controller che l'approvazione o il rifiuto di una richiesta di tirocinio curriculare da parte del tutor accademico e del referente aziendale nel caso di tirocinio esterno. Mentre nel caso di tirocinio interno, l'approvazione o il rifiuto spetta solo al tutor accademico.
visualizzaProgettoFormativo	Controller che permette la visualizzazione del progetto formativo in formato PDF allo studente, al tutor accademico, al tutor aziendale e al referente aziendale, nel caso di tirocinio esterno, una volta che la richiesta di tirocinio è stata approvata. Inoltre il PdCD ha la possibilità di vedere tale progetto formativo solo e se è stato approvati dagli attori elencati.
visualizzaElencoRichiesteTirocinio	Controller che elenca la lista delle richieste di tirocinio curriculare compilati dagli studenti.

6.1.4 Gestione Tirocinio Curriculare

Classe	Descrizione
visualizzaElencoTirocinanti	Controller che permette la visualizzazione dell'elenco dei tirocinanti del dipartimento nel caso dell'ufficio Carriere. Nel caso dell'azienda, permette la visualizzazione dell'elenco dei propri tirocinanti la cui hanno concordato con l'azienda stessa. Nel caso del tutor accademico e del tutor aziendale, permette la visualizzazione dell'elenco dei propri tirocinanti. Mentre per lo studente permette di visualizzare i propri tirocini.
visualizzaElencoAziende	Controller che permette la visualizzazione dell'elenco delle aziende convenzionate con delle informazioni sull'azienda rispettiva e le proposte di tirocinio da parte dell' utente ospite e dell'utente registrato.
visualizzaElencoDocenti	Controller che permette la visualizzazione dell'elenco dei docenti disponibili, ognuno con le varie proposte di tirocinio

	curriculare interno, da parte dell'utente ospite e dell'utente non registrato.
compilaRelazione	Controller che gestisce la relazione compilata, in cui vi contiene una descrizione e l'esito del tirocinio svolto da uno studente, dal tutor aziendale solamente nel tirocinio esterno.
compilaQuestionario	Controller che gestisce il questionario, dove quest'ultimo contiene delle informazioni atte a migliorare il tirocinio, compilato da parte del tirocinante e del tutor aziendale nella quale si verifica solamente nel tirocinio esterno.
valutaRelazione	Controller che gestisce l'approvazione o il rifiuto di un verbale del termine tirocinio da parte del tutor accademico. Ciò si verifica solamente nel tirocinio esterno.
visualizzaRelazione	Controller che permette la visualizzazione di un verbale includenti dei dettagli specifici inerenti al tirocinio terminato da uno studente.
visualizzaQuestionario	Controller che permette la visualizzazione di un questionario includenti delle informazioni atti a migliorare il tirocinio.
visualizzaInfomazioniAzienda	Controller che permette la visualizzazione dei dettagli specifici di un'azienda.

6.1.5 Gestione Registro Tirocinio Curriculare

Classe	Descrizione
visualizzaRegistro	Controller che permette la visualizzazione il registro di tirocinio e i relativi dettagli da parte dello studente, del tutor accademico e del tutor aziendale.
compilaRegistro	Controller che permette allo studente la compilazione del registro di tirocinio ogniqualvolta viene compiuta un'attività.
valutaAttività	Controller che gestisce l'approvazione o il rifiuto di un'attività compilata dallo studente da parte del tutor accademico e dal Referente Aziendale nel caso di tirocinio esterno. Mentre nel caso di

	tirocinio interno l'approvazione o il rifiuto spetta solo al tutor accademico.
valutaRegistro	Controller che gestisce l'approvazione o il rifiuto del registro di tirocinio di uno studente da parte del PdCD e dal tutor accademico nel caso di tirocinio esterno. Mentre per il tirocinio esterno, l'approvazione o il rifiuto spetta soltanto al PdCD.
visualizzaAttività	Controller che permette la visualizzazione delle informazioni inerenti all'attività selezionata.

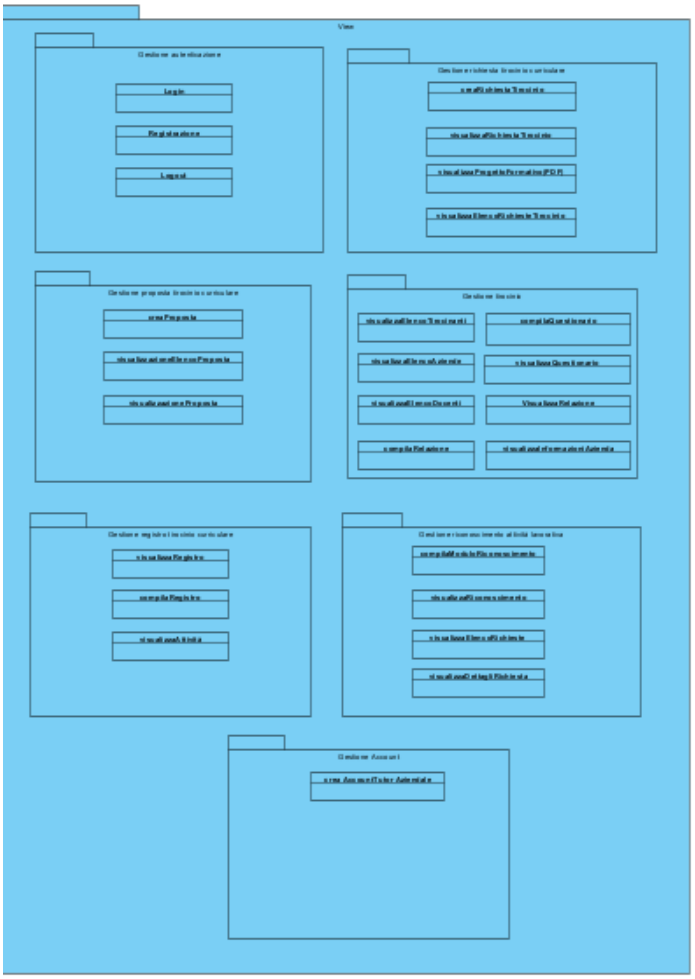
6.1.6 Gestione Riconoscimento Attività Lavorativa

Classe	Descrizione
compilaModuloRiconoscimento	Controller che gestisce il modulo compilato dallo studente che intende validare il riconoscimento attività lavorativa.
visualizzaRiconoscimento	Controller che permette la visualizzazione del modulo compilato, i relativi dettagli e lo stato della richiesta nel caso dello studente. Nel caso del PdCD, permette la visualizzazione un elenco di richieste del riconoscimento attività lavorativa di tutti gli studenti che hanno sottomesso il proprio modulo. Mentre nel caso dell'Ufficio Carriere, permette la visualizzazione un elenco di tutti gli studenti che hanno sottomesso il proprio modulo.
valutaRiconoscimento	Controller che permette l'approvazione o il rifiuto di una richiesta del modulo di riconoscimento attività lavorativa di uno studente da parte del PdCD.
visualizzaElencoRichieste	Controller che elenca la lista delle richieste di riconoscimento attività lavorativa compilati dagli studenti.
visualizzaDettagliRichiesta	Controller che permette la visualizzazione delle informazioni specifici inerenti ad una richiesta di riconoscimento attività lavorativa selezionata.

6.1.7 Gestione Account

Classe	Descrizione
creaAccountTutorAziendale	Controller che gestisce la creazione dell'account di un tutor aziendale.

6.2 Package View



6.2.1 Gestione Autenticazione

Classe	Descrizione
Login	Pagina che permette l'accesso da parte di un utente registrato nel sistema EVIM.
Logout	Pagina che conferma l'uscita di un utente dal sistema EVIM.
Registrazione	Pagina che permette all'utente di registrarsi sulla piattaforma EVIM

6.2.2 Gestione Proposta Tirocinio Curriculare

Classe	Descrizione
creaProposta	Pagina che permette la creazione di una proposta di tirocinio da parte del tutor accademico e dall'azienda.
visualizzaElencoProposta	Pagina che visualizza le proposte di tirocinio nel caso del tutor accademico, tutor aziendale. Mentre nel caso del referente aziendale, permette la visualizzazione di tutte le proposte di tirocinio da lui pubblicate.
VisualizzazioneProposta	Pagina che permette di visualizzare nel dettaglio la proposta pubblicata.

6.2.3 Gestione Richiesta Tirocinio Curriculare

Classe	Descrizione
creaRichiestaTirocinio	Pagina che permette di inserire le informazioni necessarie come la selezione del Tutor accademico (per il Tirocinio Interno esterno) o l'azienda convenzionata (per il Tirocinio Esterno), la proposta di tirocinio e l'inoltro della richiesta.
visualizzaRichiestaTirocinio	Pagina che permette la visualizzazione delle richieste di tirocinio inoltrate dagli studenti nel caso del tutor accademico e del referente aziendale. Nel caso dello studente, permette la visualizzazione le proprie richieste di tirocinio inoltrate.Inoltre permette anche la conferma o rifiuto da parte del PdCD della stessa
visualizzaProgettoFormativo	Pagina che permette la visualizzazione del progetto formativo in formato PDF allo studente, al tutor accademico, al tutor aziendale e al referente aziendale, nel caso di tirocinio esterno, una volta che la richiesta di tirocinio è stata approvata. Inoltre il PdCD ha la possibilità di vedere

	tale progetto formativo solo e se è stato approvati dagli attori elencati.

6.2.4 Gestione Tirocinio Curriculare

Classe	Descrizione
visualizzaElencoTirocinanti	Pagina che permette la visualizzazione dell'elenco dei tirocinanti del dipartimento nel caso dell'ufficio Carriere. Nel caso dell'azienda, permette la visualizzazione dell'elenco dei propri tirocinanti la cui hanno concordato con l'azienda stessa. Nel caso del tutor accademico e del tutor aziendale, permette la visualizzazione dell'elenco dei propri tirocinanti. Mentre per lo studente permette di visualizzare i propri tirocini.
visualizzaElencoAziende	Pagina che permette la visualizzazione dell'elenco delle aziende convenzionate con delle informazioni sull'azienda rispettiva e le proposte di tirocinio da parte dell' utente ospite e dell'utente registrato.
visualizzaElencoDocenti	Pagina che permette la visualizzazione dell'elenco dei docenti disponibili, ognuno con le varie proposte di tirocinio curriculare interno, da parte dell'utente ospite e dell'utente non registrato.
compilaRelazione	Pagina che visualizza il form riguardante la relazione da compilare, in cui vi contiene un descrizione e l'esito del tirocinio svolto da uno studente, dal tutor aziendale solamente nel tirocinio esterno.
compilaQuestionario	Pagina che visualizza il form del questionario,dove quest'ultimo contiene delle informazioni atti a migliorare il tirocinio, compilato da parte del tirocinante e del tutor aziendale nella quale si verifica solamente nel tirocinio esterno.
visualizzaQuestionario	Pagina che permette la visualizzazione del questionario precedentemente compilato dallo Studente

visualizzaRelazione	Pagina che visualizza il verbale del termine tirocinio da parte del tutor accademico. Ciò si verifica solamente nel tirocinio esterno.
VisualizzaInformazioniAzienda	Pagina che permette la visualizzazione delle informazioni dell'azienda

6.2.5 Gestione Registro Tirocinio Curriculare

Classe	Descrizione
visualizzaRegistro	Pagina che permette la visualizzazione il registro di tirocinio e i relativi dettagli da parte dello studente, del tutor accademico e del tutor aziendale.
compilaRegistro	Pagina che permette allo studente la visualizzazione del form per la compilazione del registro di tirocinio ogniqualvolta viene compiuta un'attività.
visualizzaAttività	Pagina che gestisce l'approvazione o il rifiuto di un'attività compilata dallo studente da parte del tutor accademico e dal Referente Aziendale nel caso di tirocinio esterno. Mentre nel caso di tirocinio interno l'approvazione o il rifiuto spetta solo al tutor accademico.

6.2.6 Gestione Riconoscimento Attività Lavorativa

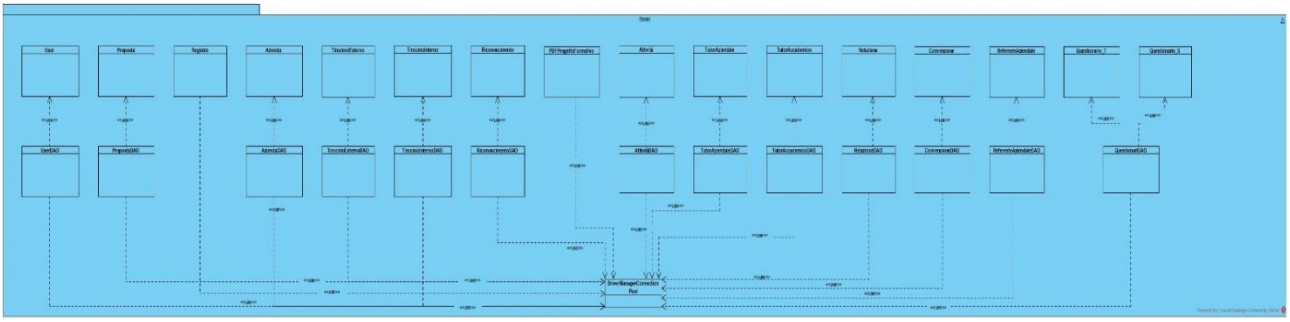
Classe	Descrizione
compilaModuloRiconoscimento	Pagina che visualizza il form che dovrà essere compilato dallo studente che intende validare il riconoscimento attività lavorativa.
visualizzaRiconoscimento	Pagina che permette la visualizzazione del modulo compilato, i relativi dettagli e lo stato della richiesta nel caso dello studente. Nel caso del PdCD, permette la visualizzazione un elenco di richieste del riconoscimento attività lavorativa di tutti gli studenti che hanno sottomesso il proprio modulo. Mentre nel caso dell'Ufficio Carriere,

	permette la visualizzazione un elenco di tutti gli studenti che hanno sottomesso il proprio modulo.
visualizzaElencoRichieste	Pagina che permette la visualizzazione dell'elenco delle richieste sottomesse di uno studente da parte del PdCD.
visualizzaDettagliRichiesta	Pagina che permette al PdCD di visualizzazione i dettagli della richiesta di riconoscimento da uno student. Inoltre permette anche il rifiuto o la conferma della richiesta associata.

6.2.7 Gestione Account

Classe	Descrizione
CreaAccountTutorAziendale	Pagina che visualizza il form che dovrà essere compilato dall'azienda, con tutti i dati relativi al tutor aziendale associato.

6.3 Package Model



Classe	Descrizione
User	Model che rappresenta tutti gli utenti che possono utilizzare sia i servizi inerenti alla piattaforma EV che IM, cioè Tirocinante, PdCD e Ufficio Carriere.
UserDAO	Model che permette di eseguire le operazioni sul db per la tabella User.
Proposta	Model che rappresenta le informazioni inerenti alla proposta di tirocinio.
PropostaDAO	Model che permette di eseguire le operazioni sul db per la tabella Proposta.
Registro	Model che rappresenta le informazioni del Registro di tirocinio e un insieme di attività di tirocinio.
Azienda	Model che rappresenta le informazioni dell'Azienda.
AziendaDAO	Model che permette di eseguire le operazioni sul db per la tabella Azienda.
TirocinioEsterno	Model che rappresenta le informazioni inerenti al Tirocinio esterno.
TirocinioEsternoDAO	Model che permette di eseguire le operazioni sul db per la tabella TirocinioEsterno.
TirocinioInterno	Model che rappresenta le informazioni inerenti al Tirocinio interno.

TirocinioInternoDAO	Model che permette di eseguire le operazioni sul db per la tabella TirocinioInterno.
Riconoscimento	Model che rappresenta le informazioni inerenti al modulo di riconoscimento attività lavorativa.
RiconoscimentoDAO	Model che permette di eseguire le operazioni sul db per la tabella Riconoscimento.
Attività	Model che racchiude le informazioni inerenti ad una singola attività di tirocinio.
AttivitàDAO	Model che permette di eseguire le operazioni sul db per la tabella Attività.
TutorAziendale	Model che racchiude le informazioni del Tutor Aziendale.
TutorAziendaleDAO	Model che permette di eseguire le operazioni sul db per la tabella TutorAziendale.
TutorAccademico	Model che racchiude le informazioni del Tutor Accademico.
TutorAccademicoDAO	Model che permette di eseguire le operazioni sul db per la tabella TutorAccademico.
Relazione	Model che racchiude le informazioni della Relazione di Termine tirocinio.
RelazioneDAO	Model che permette di eseguire le operazioni sul db per la tabella Relazione.
Convenzione	Model che racchiude le informazioni della convezione stipulata da un'azienda con l'Università degli Studi di Salerno per lo svolgimento di Tirocini Esterni.
ConvenzioneDAO	Model che permette di eseguire le operazioni sul db per la tabella Convenzione.
DriverManagerConnectionPool	Model la cui istanza rappresenta la connessione con il database, per ulteriori dettagli vedere Design Pattern Singleton nelle sezioni precedenti.
PDFProgettoFormativo	Model che rappresenta l'astrazione del documento pdf che rappresenta un

	progetto formativo.
ReferenteAziendale	Model che racchiude le informazioni del Referente Aziendale.
ReferenteAziendaleDAO	Model che permette di eseguire le operazioni sul db per la tabella ReferenteAziendale.
Questionario_S	Model che rappresenta il questionario valutativo che uno studente deve compilare al termine dello svolgimento di un Tirocinio Curriculare.
Questionario_T	Model che rappresenta il questionario che un tutor deve compilare al termine dello svolgimento di un Tirocinio Curriculare.
QuestionariDAO	Model che permette di eseguire le operazioni sul db per le tabella Questionario_S e Questionario_T.

7. Class Interfaces

7.1 Class Interface Controller

1.1.1 Gestione Autenticazione

2. Nome Classe
Login
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente non deve essere loggato nella piattaforma "EVIM". L'utente deve essere presente sul database affinché vada a buon fine il login.

Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):l'utente si è loggato al sistema "EVIM".

Nome Classe
Logout
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente deve essere loggato nella piattaforma "EVIM".
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):l'utente risulta disconnesso dal sistema "EVIM".

Nome Classe
Registrazione
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet()/doPost(): l'utente non deve essere registrato nella piattaforma "EVIM".
Post-condizione
doGet()/doPost():l'account appena creato è inserito con successo nel database.

6.1.2 Gestione Proposta Tirocinio

7 Nome Classe
creaProposta
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la proposta di tirocinio creata non esiste nel database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):l'inserimento della proposta di tirocinio nel database è andata a buon fine

Nome Classe
visualizzaElencoProposta
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): le proprie proposte di tirocinio curriculare devono essere presenti sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca la lista delle proposte di tirocinio curriculare

Nome Classe
visualizzaDettagliProposta
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
-
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca i dettagli della proposta selezionata.

6.1.3 Gestione Richiesta Tirocinio

7 Nome Classe
creaRichiestaTirocinio
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la richiesta di tirocinio creata non deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'inserimento della richiesta di tirocinio nel database è andata a buon fine

Nome Classe
visualizzaRichiestaTirocinio
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): le richieste di tirocinio devono essere presenti sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca la richiesta di tirocinio selezionata

Nome Classe
valutaRichiesta
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void

+doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente deve essere loggato nella piattaforma EVIM
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la modifica dello stato della richiesta sul database avviene a buon fine

Nome Classe
visualizzaProgettoFormativo
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): i progetto formativi devono essere presenti sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca il progetto formativo selezionato.

Nome Classe
valutaProgettoFormativo
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void

Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente deve essere loggato nella piattaforma EVIM
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la modifica dello stato del progetto formativo sul database avviene a buon fine

Nome Classe
visualizzaElencoRichiesteTirocinio
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): le richieste di tirocinio devono essere presenti sul database.
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):il sistema elenca la lista delle richieste di tirocinio.

6.1.4 Gestione Tirocinio

7 Nome Classe
visualizzaElencoTirocinanti
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void

+doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):i tirocinanti devono essere presenti sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca la lista dei tirocinanti

Nome Classe
visualizzaElencoAziendeServlet
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):le aziende devono essere presenti sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca la lista delle aziende disponibili

Nome Classe
visualizzaElencoDocenti
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void

+doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):i docenti devono essere presenti sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca la lista dei docenti disponibili

Nome Classe
compilaRelazione
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la relazione non deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la relazione viene compilata e inserita nel database con successo

Nome Classe
compilaQuestionario
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void

+doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il questionario non deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il questionario viene compilata e inserita nel database con successo

Nome Classe
valutaRelazione
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente deve essere loggato sulla piattaforma EVIM
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la modifica dello stato della relazione sul database avviene a buon fine

Nome Classe
visualizzaRelazione
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void

+doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la relazione deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca i dettagli della relazione

Nome Classe
visualizzaQuestionario
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il questionario deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca i dettagli del questionario

Nome Classe
visualizzaInformazioniAzienda
Attributi
-
Metodi

+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
-
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca i dettagli dell'azienda selezionata.

6.1.4 Gestione Registro Tirocinio

Nome Classe
visualizzaRegistro
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il registro deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca il registro

Nome Classe
compilaRegistro
Attributi
-
Metodi

+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il registro non deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il registro viene compilata e inserita con successo nel database

Nome Classe
valutaAttività
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente deve essere registrato sulla piattaforma EVIM
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la modifica dello stato dell'attività sul database avviene a buon fine

Nome Classe
valutaRegistro
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void

Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente deve essere registrato sulla piattaforma EVIM
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la modifica dello stato del registro sul database avviene a buon fine

Nome Classe
visualizzaAttività
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'attività deve essere presente sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca i dettagli dell'attività selezionata.

6.1.5 Gestione Riconoscimento attività lavorativa

Nome Classe
compilaModuloRiconoscimento
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void

+doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response):il modulo di riconoscimento attività lavorativa non deve essere presente sul database.
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'inserimento del modulo nel database va a buon fine

Nome Classe
visualizzaRiconoscimento
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
-
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca il modulo di riconoscimento attività lavorativa

Nome Classe
valutaRichiestaRiconoscimento
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void

Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'utente deve essere registrato sulla piattaforma EVIM
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): la modifica dello stato del modulo di riconoscimento attività lavorativa sul database avviene a buon fine

Nome Classe
visualizzaElencoRichieste
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): le richieste di riconoscimento attività lavorativa devono essere presenti sul database
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca la lista delle richieste di riconoscimento attività lavorativa

Nome Classe
visualizzaDettagliRichieste
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void

Pre-condizioni
-
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): il sistema elenca i dettagli della richiesta di riconoscimento attività lavorativa selezionata

6.1.6 Gestione Account

Nome Classe
creaAccountTutorAziendale
Attributi
-
Metodi
+doGet(HttpServletRequest request, HttpServletResponse response):void +doPost(HttpServletRequest request, HttpServletResponse response):void
Pre-condizioni
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'account non deve essere presente sul database.
Post-condizione
doGet(HttpServletRequest request, HttpServletResponse response)/doPost(HttpServletRequest request, HttpServletResponse response): l'account appena creato viene inserito a buon fine nel database.

6.2 Class Interface Model

Nome Classe
User
Attributi
- email : String - name : String - surname : String

- sex : char - password : String - userType : int - corso : String - luogoDiNascita : String - dataDiNascita : String - residente : String - via : String - telefono : String - matricola : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
UserDAO
Attributi
NA
Metodi
+ doRetrieveByLoginData (int usertype, String email, String password) : User + insertNewUser (User u) : boolean
Pre-condizioni
doRetrieveByLoginData(int, String, String) : email e password devono corrispondere a una coppia presente sul db insertNewUser (User) : u deve essere un utente non presente sul db.
Post-condizione
doRetrieveByLoginData(int, String, String) : User deve essere un utente presente nel db identificato dalla coppia username e password passati come parametri. insertNewUser (User u) : boolean deve essere true se l'inserimento va a buon fine, false altimenti.

Nome Classe
Proposta
Attributi
- ID_Proposta: int - ID_Azienda : int - ID_Tutor : int - obiettivi : String - competenze : String - attività : String - modalità : String - nomeTutorAziendale : String - cognomeTutorAziendale : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
PropostaDAO
Attributi
NA
Metodi
+ findByIdAzienda (int id_azienda) : ArrayList<Proposta> + getListaProposta( ) : ArrayList<Proposta>

+ findByIdTutorAccademico (int idTutorAccademico) : ArrayList<Proposta> + findByIdTutorAziendale (int idTutorAziendale) : ArrayList<Proposta> + getProposteAziendaWithIdAzienda (int idAzienda) : ArrayList<Proposta> + insertPropostaInterno (String obiettivi, String competenze, String attivita, String modalita, int idTutorAccademico) : boolean + insertPropostaEsterno (String obiettivi, String competenze, String attivita, String modalita, int idAzienda, int intTutorAziendale) : boolean + getPropostaInterno (int idProposta) : Proposta + getPropostaEsterno (int idProposta) : Proposta + modificationPropostaEsterno (String obiettivo, String competenze, String attivita, String modalita, int tutorAziendale, int idProposta) : boolean + modificationPropostaInterno (String obiettivo, String competenze, String attivita, String modalita, int idProposta) : boolean + removeProposta (int idProposta) : boolean
Pre-condizioni
findByIdAzienda (int) : id_azienda deve essere un id di un'azienda presente sul db. findByIdTutorAccademico(int) : idTutorAccademico deve essere un id di un tutor accademico presente sul db findByIdTutorAziendale(int) : idTutorAziendale deve essere un id di un tutor aziendale presente sul db getProposteAziendaWithIdAzienda (int) : idAzienda deve essere un id di un'azienda presente sul db insertPropostaInterno(String, String, String, String, int)/insertPropostaEsterno(String, String, String, String, int, int) : idTutorAccademico /idAzienda e idTutorAziendale deve/ono essere id presente/i sul db. getPropostaInterno(int)/getPropostaEsterno(int)/removeProposta(int) : idProposta deve essere l'identificativo di una proposta presente sul db.
Post-condizione
findByIdAzienda (int id_azienda) : ArrayList<Proposta> deve contenere tutte le proposte dell'azienda con identificativo id_azienda.

getListaProposta( ) : ArrayList<Proposta> deve contenere tutte le proposte di tirocinio presenti sul db.

findByIdTutorAccademico (int idTutorAccademico) : ArrayList<Proposta> deve contenere tutte le proposte di tirocinio del tutor accademico con identificativo idTutorAccademico

findByIdTutorAziendale (int idTutorAziendale) : ArrayList<Proposta> deve contenere tutte le proposte di tirocinio del tutor aziendale con identificativo idTutorAziendale

getProposteAziendaWithIdAzienda (int idAzienda) : ArrayList<Proposta> deve contenere tutte le proposte di tirocinio dell'azienda con identificativo idAzienda

insertPropostaInterno (String obiettivi, String competenze, String attivita, String modalita, int idTutorAccademico) : boolean deve essere true se l'inserimento della nuova tupla all'interno del database è andata a buon fine, false altrimenti.

insertPropostaEsterno (String obiettivi, String competenze, String attivita, String modalita, int idAzienda, int intTutorAziendale) : boolean deve essere true se l'inserimento della nuova tupla all'interno del database è andata a buon fine, false altrimenti.

getPropostaInterno (int idProposta) : Proposta deve corrispondere alla tupla presente nel database con chiave primaria idProposta

getPropostaEsterno (int idProposta) : Proposta deve corrispondere alla tupla presente nel database con chiave primaria idProposta

modificationPropostaEsterno (String obiettivo, String competenze, String attivita, String modalita, int tutorAziendale, int idProposta) : boolean deve essere true se la tupla nel database identificata da idProposta è stata modificata correttamente, false altrimenti.

modificationPropostaInterno (String obiettivo, String competenze, String attivita, String modalita, int idProposta) : boolean deve essere true se la tupla nel database identificata da idProposta è stata modificata correttamente, false altrimenti.

removeProposta (int idProposta) : boolean deve essere true se la proposta identificata dalla chiave primaria idProposta è stata rimossa con successo dal database, false altrimenti.

Nome Classe
Registro
Attributi

- ID_Registro : int - TirocinioEsterno : int - FirmaResponsabile : int - FirmaTutorAccademico : int - Status : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
RegistroDAO
Attributi
- NA
Metodi
+ doSave (Integer ID_Tirocinio, String status, String tipo) : int + doAlterFirmaTutorAc (int idregistro) : boolean + doAlterFirmaTutorAz (int idregistro) : boolean + doAlterFirmaTutorInterno (int idregistro) : boolean + doAlterFirmaPdcdEsterno (int idtirocinio) : boolean + doAlterFirmaPdcdInterno (int idtirocinio) : boolean
Pre-condizioni
doSave(Integer, String, String) : ID_Tirocinio è un id di un tirocinio presente sul db.
Post-condizione
doSave(Integer, String, String) : int è > 0 se l'update è andato a buon fine, 0 altrimenti.

Nome Classe
Azienda
Attributi
- idAzienda : int - cf : String - telefono : String - nome : String - password : String - email: String - sitoWeb: String - indirizzo: String - descrizione: String - numeroDipendenti: String - codiceAteco: String - idReferente: String - idConvenzione: String - proposte: ArrayList<Proposta>
Metodi
+getters&setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
AziendaDAO
Attributi
NA
Metodi
+ doRetriveAll( ): ArrayList<Azienda> + findByID(Integer id): Azienda

+ doRetriveByLoginData(String email, String password) : Azienda
Pre-condizioni
findByID(Integer) : id deve essere un id di un'azienda presente nel db. doRetriveByLoginData(String, String) : email e password devono essere corrispondenti nel db.
Post-condizione
doRetriveAll( ): ArrayList<Azienda> deve contenere tutte le aziende presenti nel db. findByID(Integer id): Azienda deve contenere tutti i campi della tupla presente nel db identificata dalla chiave primaria id doRetriveByLoginData(String, String) : Azienda deve contenere tutti i campi della tupla presente nel db identificata dalla coppia email e password inseriti come parametri.

Nome Classe
TirocinioEsterno
Attributi
- ID_TirocinioEsterno : int - ID_TutorAccademico : int - ID_TutorAziendale : int - OreTotali : int -CFU : int -ID_Proposta : int - EMAIL : String - data : String - status : String - FirmaAziendale : Boolean - FirmaTutorAziendale : Boolean - FirmaTutorAccademico : Boolean - FirmaPdCD : Boolean
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione

NA

Nome Classe
TirocinioEsternoDAO
Attributi
NA
Metodi
+ doInsert (int idTutAcc, int idTutAz, int oreTotali, int cfu, int idProposta, String email, String data, String status) : boolean + doRetriveAllByStudent (String EMAIL) : ArrayList <TirocinioQueryEsternoStudente> + doRetriveAllByStudentRegistro (String EMAIL) : ArrayList <TirocinioEsterno> + doRetriveAllValutazionePdCD ( ) : ArrayList <TirocinioQueryPdCD> + doRetriveAllByTutorAcc (int IDTutor) : ArrayList<TirocinioQueryEsternoTutorAcc> + updateFirmaTrueAzienda (boolean b, int idTirocinio, int idAzienda) : int + updateFirmaFalseAzienda (boolean b, int idTirocinio, int id) : int + getProgettoFormativoEsterno (int id) : PDFProgettoFormativo + updateFirmaTureTutorAccademico (boolean firma, int idTirocinio, int idTutorAccademico) : int + updateFirmaPdCDTrue (boolean firma, int idTirocinio) : int + updateFirmaPdCDFalse (boolean firma, int idTirocinio) : int + updateFirmaFalseTutorAccademico (boolean firma, int idTirocinio, int idTutorAccademico) : int + updateFirmaTrueAziendale (boolean b, int idTirocinio, int idTutorAziendale) : int + updateFirmaFalseAziendale (boolean b, int idTirocinio, int id) : int + doRetriveTirocinioInSvolgimentoStudente (String EMAIL) : ArrayList<TirocinioEsterno> + doRetriveTirocinioInSvolgimentoStudenteRegistro (String email) :

ArrayList<RegistroQuery> + doRetriveTirocinioInSvolgimentoTutorAccRegristro (int id) : ArrayList<RegistroQuery> + doRetriveTirocinioInSvolgimentoTutorAzRegistro (int id) : ArrayList<RegistroQuery> + doRetriveTirocinioInSvolgimentoPdcdRegistro ( ) : ArrayList<RegistroQuery> + doRetriveAllByTutorAz (int id) : ArrayList<TirocinioQueryEsternoTutorAz> + doRetriveAllByAzienda (int id) : ArrayList<TirocinioQueryEsternoTutorAz> + getStatusTirocinioEsterno (String email) : boolean
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
TirocinioInterno
Attributi
- ID_TirocinioInterno : int - ID_tutorAccademico : int - OreTotali : int - numeroCFU : int - ID_Proposta : int - EMAIL : String - status : String - FrimaPdCD : Boolean - FirmaTutorAccademico : Boolean - data : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione

NA

Nome Classe
TirocinioInternoDAO
Attributi
NA
Metodi
+ doInsert (String email, int idTutAcc, String data, int oreTot, String status, int numCFU, int IDProp) : boolean + doRetriveAllByStudent (String EMAIL) : ArrayList<TirocinioQueryInternoStudente> + doRetriveAllByTutorAcc (int IDTutor) : ArrayList<TirocinioQueryInternoTutorAcc> + getProgettoFormativoInterno (int id) : PDFProgettoFormativo + doRetriveAllValutazionePdCD ( ) : ArrayList<TirocinioQueryPdCD> + updateFirmaTrue (boolean firma, int idTirocinio, int idTutorAccademico) : int + updateFirmaFalse (boolean firma, int idTirocinio, int idTutorAccademico) : int + doRetriveTirocinioInSvolgimentoStudenteRegistro (String email) : ArrayList<RegistroQuery> + updateFirmaPdCDTrue (boolean firma, int idTirocinio) : int + updateFirmaPdCDFalse (boolean firma, int idTirocinio) : int + doRetriveTirocinioInSvolgimentoTutorAccRegistro (int id) : ArrayList<RegistroQuery> + doRetriveTirocinioInSvolgimentoPdcdRegistro ( ) : ArrayList<RegistroQuery> + getStatusTirocinioInterno (String email) : boolean
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
Riconoscimento
Attributi
- idRiconoscimento : int - emailUser : String - enteAzienda : String - profilo : String - indirizzoSede : String - tipoContratto : String - periodo : String - oreSvolte : int - CFUTirocinioEsterno : int - CFUTirocinioObbligatorio : int - CFUAccompagnamentoLavoro : int - stato : String - nomeStudente : String - cognomeStudente : String - matricolaStudente : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
RiconoscimentoDAO
Attributi
NA
Metodi
+ insertRiconoscimento (String emailUser, String enteAzienda, String indirizzoSede, String profilo, String tipoContratto, String periodo, int oreSvolte, int CFUTirocinioObbligatorio, int CFUTirocinioEsterno, int CFUAccompagnamento) : boolean + getModuloRiconoscimento (String emailUser) : Riconoscimento

+ getModuliRiconoscimentiWithStudenti ( ) : ArrayList<Riconoscimento> + getModuliRiconoscimentiWithEmailStudente (String emailStudente) : ArrayList<Riconoscimento> changeStatoModulo (int idRiconoscimento, String modifica) : boolean
Pre-condizioni
insertRiconoscimento (String , String , String , String , String , String , int , int , int , int) : emailUser deve essere una chiave presente nel db. getModuloRiconoscimento (String) : emailUser deve corrispondere ad una email di uno studente registrato nel db getModuliRiconoscimentiWithEmailStudente(String) : emailStudente deve corrispondere ad una email di uno studente registrato nel db. changeStatoModulo(int, String) : idRiconoscimento deve corrispondere ad un id di un modulo di riconoscimento presente nel db.
Post-condizione
insertRiconoscimento (String , String , String , String , String , String , int , int , int , int) : boolean true se l'inserimento di un nuovo modulo di riconoscimento sul db è andato a buon fine, false altrimenti. getModuloRiconoscimento (String) : Riconoscimento deve corrispondere ad un riconoscimento presente sul db. getModuliRiconoscimentiWithStudenti() : ArrayList<Riconoscimento > Deve contenere tutti i riconoscimenti presenti nel db. getModuliRiconoscimentiWithEmailStudente(String) : ArrayList<Riconoscimento> Deve contenere tutti i mdouli presenti nel db associati ad uno studente identificato dall'email passata come parametro. changeStatoModulo(int idRiconoscimento, String modifica) : boolean true se il campo state del modulo di riconoscimento identificato dalla chiave primaria idRiconoscimento presente nel db è stato modificato con la stringa modifica, false se la modifica non è andata a buon fine.

Nome Classe

Attività
Attributi
- ID_Attivita: int - ID_Registro: int - Descrizione: String - OrarioIngresso: int - OrarioUscita: int - FirmaResponsabile: boolean
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
AttivitàDAO
Attributi
NA
Metodi
+ doInsert (int idRegistro, String descrizione, int orarioIngresso, int orarioUscita) : int + doRetriveAllInternoTutorAcc (String email, int idTutorAcc) : ArrayList <RegistroQuery> + doRetriveAllEsternoTutorAcc (String email, int iDTutorAcc) : ArrayList <RegistroQuery> + doRetriveAllEsternoTutorAzi (int iDTurorAziendale, String email) : ArrayList <RegistroQuery> + doRetriveAllInterno(String email) : ArrayList <RegistroQuery> + doRetriveAllEsterno (String email) : ArrayList <RegistroQuery> + doRetriveAllEsternoPdCD( ) : ArrayList<RegistroQuery>

+ doRetriveAllInternoPdCD( ) : ArrayList<RegistroQuery>
Pre-condizioni
NA
Post-condizione
doInsert(int, String, int, int): viene creata una nuova tupla corrispondente ad una nuova attività. doRetriveAllInternoTutorAcc(int): ArrayList<Attività> restituito contiene tutte le attività di tirocinio interno fatte da uno studente che ha come tutor accademico il tutor con chiave primaria i. doRetriveAllEsternoTutorAcc(int): ArrayList<Attività> restituito contiene tutte le attività di tirocinio esterno fatte da uno studente che ha come tutor accademico il tutor con chiave primaria i. doRetriveAllEsternoTutorAzi (int): ArrayList<Attività> restituito contiene tutte le attività di tirocinio esterno fatte da uno studente che ha come tutor aziendale il tutor con chiave primaria i. doRetriveAllEsterno(String)/doRetriveAllInterno(String) : ArrayList<Attività> restituito contiene tutte le attività di tirocinio esterno/interno fatte da uno studente che ha come chiave primaria email.

Nome Classe
TutorAziendale
Attributi
- id : int - idAzienda : int - nome : String - cognome : String - email : String - password : String - telefono : String - sesso : String
Metodi
+ getters & setters
Pre-condizioni
NA

Post-condizione
NA

Nome Classe
TutorAziendaleDAO
Attributi
NA
Metodi
+ doRetriveByIDProposta(int IDProp) : int + doRetrieveByLoginData(String email, String password) : TutorAziendale + getElencoTutorAziendali (int idAzienda) : ArrayList<TutorAziendale> + getInformationTutorAziendale (int idTutorAziendale) : TutorAziendale + doSave (int ID_Azienda, String nome, String cognome, String email, String password, String telefono) : int
Pre-condizioni
doRetriveByIDProposta(int) : IDProp Deve essere un identificativo di una proposta presente nel database doRetrieveByLoginData(String, String) : email e password devono essere corrispondenti nel database getElencoTutorAziendali(int) : idAzienda deve essere l'id di un azienda presente nel db getInformationTutorAziendale(int) : idTutorAziendale deve essere l'id di un tutor aziendale presente nel db
Post-condizione
doRetriveByIDProposta(int IDProp) : int è l'id del tutor aziendale associato alla proposta identificata da IDProp, -1 in caso di proposta non trovata. doRetrieveByLoginData(String email, String password) : TutorAziendale deve contenere i dati corrispondenti della tupla presente nel db identificata da email e password.

getElencoTutorAziendali (int idAzienda) : ArrayList<TutorAziendale> deve contenere tutti i tutor aziendali dell'azienda identificata sul db con identificiativo idAzienda getInformationTutorAziendale (int idTutorAziendale) : TutorAziendale deve contenere i dati corrispondenti alla tupla presente nel db identificata da idTutorAziendale

Nome Classe
TutorAccademico
Attributi
- idTutorAccademico : int - nome : String - cognome : String - password : String - sesso : String - email : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
TutorAccademicoDAO
Attributi
NA
Metodi
+ doRetrieveAll ( ) : ArrayList<TutorAccademico> + doRetrieveByLoginData (String email, String password) : TutorAccademico + insertNewTutorAccademico (String nome, String cognome, String password, String sesso, String email) : boolean
Pre-condizioni
doRetrieveByLoginData(String, String) : email e password devono essere presenti nel db.
Post-condizione
doRetrieveAll ( ) : ArrayList<TutorAccademico> deve contenere tutti i tutor accademici presenti nel db. doRetrieveByLoginData (String, String) : TutorAccademico deve contenere i dati corrispondenti a quelli presenti nel db. insertNewTutorAccademico (String, String, String, String, String) : boolean è true se il nuovo tutor accademico è stato inserito in maniera corretta nel db, false altrimeti.

Nome Classe
Relazione
Attributi
- idrelazione : int - descrizione : String - email : String - status : String - idtutor : int
Metodi
+ getters & setters
Pre-condizioni

NA
Post-condizione
NA

Nome Classe
RelazioneDAO
Attributi
NA
Metodi
+ insertRelazione (int idTutor, String email, String descrizione, String status) : boolean + doRetriveStudenti (int idTutor) : ArrayList<User> + doRetriveRelazionefromId (int idRelazione) : Relazione + doAlterRelazione (boolean approva, int idrelazione) : boolean
Pre-condizioni
insertRelazione(int, String, String, String) , doRetriveStudenti(int): idTutor deve corrispondere ad un id di un tutor presente nel db doRetriveStudenti (int) : idTutor deve corrispondere all'identificativo di un tutor presente nel db doRetriveRelazionefromId (int) : idRelazione deve corrispondere ad un id di una relazione presente nel db doAlterRelazione (boolean, int) : idrelazione deve corrispondere ad un id di una relazione presente nel db
Post-condizione
insertRelazione(int, String, String, String) : boolean deve essere true se l'inserimento è andato a buon fine, false altrimenti. doRetriveStudenti (int) : ArrayList<User> deve contenere tutti gli studenti che hanno completato un tirocinio con un tutor identificato dal parametro in ingresso doRetriveRelazionefromId (int) : Relazione deve corrispondere alla tupla presente nel db alla tabella Relazione ed identificata dall'intero in ingresso.

doAlterRelazione (boolean, int) : boolean è true se la modifica nel db è andata a buon fine, false altrimenti.

Nome Classe
Convenzione
Attributi
- id : int - dataConvenzione : String - repertorio : String - durata : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
ConvenzioneDAO
Attributi
NA
Metodi
+ findByID (int id) : Convenzione
Pre-condizioni
findByID (int id) : id deve essere l'identificativo di una convenzione già presente sul db
Post-condizione
findByID (int id) : Convenzione ha come id l'id passato come parametro.

Nome Classe
DriverManagerConnectionPool

Attributi
- datasource : DataSource
Metodi
+ getConnection( ) : Connection
Pre-condizioni
getConnection() : Il database deve essere disponibile alla porta 3306 e le credenziali di accesso al database devono essere le stesse indicate.
Post-condizione
getConnection() : Connection deve essere una connessione attiva al database.

Nome Classe
PDFProgettoFormativo
Attributi
- nomeStudente : String - cognomeStudente : String - emailStudente : String - corsoLaurea : String - telefonoStudente : String - dataNascitaStudente : String - luogoNascitaStudente : String - residenteStudente : String - nomeDenominazione : String - sedeLegale : String - indirizzoEmail : String - codiceFiscale : String - nomeReferenteAziendale : String - cognomeReferenteAziendale : String - ruoloReferenteAziendale : String - natoReferenteAziendale : String - dataReferenteAziendale : String - codiceATECO : String - numeroDipendenti : String - nomeTutorAccademico : String - cognomeTutorAccademico : String - emailTutorAziendale : String - telefonoAziendale : String - totCFU : int - obiettivi : String - competenze : String - attivita : String - modalita : String

- totOre : int - dataConvenzione : String - reportorioConvenzione : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
ReferenteAziendale
Attributi
- codiceFiscale : String - nome : String - cognome : String - dataNascita : String - luogoNascita : String - ruolo : String
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
ReferenteAziendaleDAO
Attributi
NA
Metodi
+ findByID(String id) : ReferenteAziendale
Pre-condizioni
findByID(String id) : id deve essere un identificativo di un Referente Aziendale presente sul db
Post-condizione
findByID(String id) : ReferenteAziendale deve corrispondere alla tupla presente sul db del referenteAziendale con identificativo id.

Nome Classe
Questionario_T
Attributi
- id_questionario : int - email : String - id_tutor : int - competenze_ingresso : int - competenze_acquisite : int - utilita : int - motivazione : int - capacita : int - informazioni : int - obiettivi : int - servizi : int - assistenza : int - collaborazione : int - durata : int
Metodi
+ getters & setters

Pre-condizioni
NA
Post-condizione
NA

Nome Classe
Questionario_S
Attributi
- id_questionario : int - email : String - assistenza_disp : int - informazioni : int - servizi : int - assistenza : int - logistica : int - ambiente : int - mansioni : int - attivita : int - formazione : int - possibilita : int - valutazione : int - competenze : int
Metodi
+ getters & setters
Pre-condizioni
NA
Post-condizione
NA

Nome Classe
QuestionariDAO
Attributi
NA
Metodi
+ insertQuestionarioS (Questionario_s quest) : boolean + insertQuestionarioT (Questionario_t quest) : boolean
Pre-condizioni
insertQuestionarioS (Questionario_s) : quest deve essere un questionario non presente nel db insertQuestionarioT (Questionario_t) : quest deve essere un questionario non presente nel db
Post-condizione
insertQuestionarioS (Questionario_s quest) : boolean è true se l'inserimento nel db è andato a buon fine, false altrimenti insertQuestionarioT (Questionario_t quest) : boolean è true se l'inserimento nel db è andato a buon fine, false altrimenti